**Readme CMKV**

Nous avons décidé d’implémenter l’algorithme de Metropolis-Hastings avec recuit simulé. La raison pour laquelle cet algorithme a été choisit plutôt que l’échantillonneur de Gibbs tient au fait que ce dernier nous a semblé plus compliquer à comprendre et à implémenter.

Pour ajouter un côté artistique on a essayé d’avoir un rendu proche de l’ « ordered dithering » en utilisant des matrices de Bayer. Cela va permettre d’avoir une image qui ressemble a une image en niveau de gris bien qu’étant une image binaire.

Soit la loi de probabilité dont la densité de probabilité est . On va chercher à approximer la fonction de densité c'est-à-dire qu’on va essayer de tirer des échantillons suivant la densité. La densité peut être exprimée comme la probabilité a posteriori d’obtenir une image binaire artistique sachant que l’on dispose en entrée d’une image en niveau de gris .

On va chercher à maximiser. D’après la formule de Bayes on a :

Où est la variable aléatoire égale à l’image de sortie et est la variable aléatoire égale à l’image en entrée. est donc le produit entre la vraisemblance et la probabilité a priori, le tout divisé par un facteur de normalisation. En pratique, il est difficile de calculer ces trois probabilités. En utilisant l’algorithme de Metropolis-Hastings on peut tirer des échantillons suivant la loi de probabilité de sans en connaitre sa densité de probabilité. L’algorithme de Metropolis-Hastings que nous avons implémenté se déroule comme suit :

A chaque itération on choisit un pixel au hasard dans l’image. On choisit également un candidat au hasard suivant la loi discrète uniforme entre 0 et 1 donc soit 0 soit 1 (on veut une image en noir et blanc) :

. On va ensuite calculer le ratio des probabilités a postériori pour et à l’aide de la fonction qui est proportionnelle à la densité de :

Comme est la densité d’une loi uniforme elle est symétrique et on a :

)

On peut donc simplifier :

D’après le cours on peut choisir  comme étant:

Où est la constante de normalisation appelée fonction de partition telle que :

assure que la somme des probabilités vaut 1:

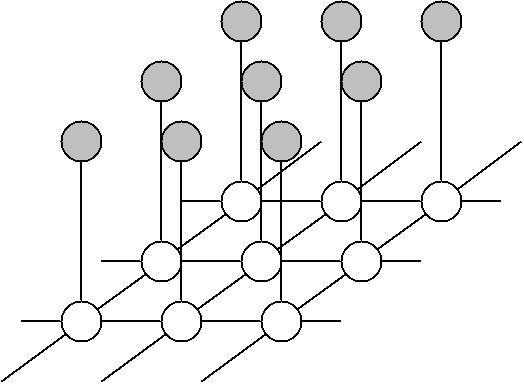
On peut donc simplifier le ratio à calculer comme ceci :

avec

étant l’énergie associée au nouvel état en acceptant et est l’énergie associée à l’état actuel (la valeur du pixel dans l’image). Le but est donc de choisir ou en fonction de la valeur minimum entre les deux énergies qui leur sont associés. Il s’agit donc d’un problème de minimisation. Si l’énergie associée à est inférieur à l’énergie associée à on choisit. Dans le cas contraire on choisit avec la probabilité. En pratique on souhaite que notre ratio ne soit pas supérieur à 1 donc doit être inférieur à 0.

Correspond à la température que l’on fixe à une valeur élevée au début de l’algorithme et que l’on diminue progressivement au cours des itérations pour obtenir une valeur de proche de zéro à la fin des itérations. Il s’agit du recuit. Au début de l’algorithme, étant grand, on va avoir une valeur de qui va être négative mais faible. Donc la valeur de va être proche de 1 ce qui aura pour conséquence de plus facilement aller vers un nouvel etat. A l’inverse, vers la fin des itérations, on aura une valeur de supérieur à zéro mais très faible ce qui implique qu’il est peu probable qu’on accepte de changer d’état. Cette mécanique est désirable car au début de l’algorithme l’image de sortie contient des valeurs qui sont éloignés du résultat recherché, on aimerait donc accepter facilement les changements d’états. A l’inverse, à la fin de l’algorithme on a des valeurs des pixels qui sont déjà proches du résultat souhaité donc on veut être restrictif concernant le changement d’état. De cette façon, il est plus facile de tomber sur un minimum global plutôt qu’on minimum local. Dans la pratique on a décidé de multiplier la température par 0.99999 à chaque itération. De cette manière en fixant la température à une valeur assez élevée de 4 au début de l’algorithme, on peut progressivement réduire la température. En faisant un million d’itérations on obtient bien une température proche de 0 à la fin de l’algorithme.

La fonction d’énergie que l’on cherche à minimiser est définie comme suit pour une valeur de pixel à l’indice :

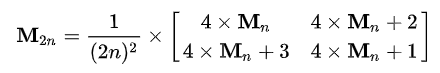


correspond à la valeur candidate ou à la valeur du pixel déjà présent dans l’image. Peut donc prendre la valeur 0 ou 1. Correspond à la valeur du pixel se situant à la même place que dans l’image en niveau de gris en entrée de la fonction ‘minimize’. correspond à la fonction indicatrice qui renvoi 1 ou vrai si est vérifiée, 0 autrement. On a donc le terme de gauche qui permet de maintenir une cohérence entre et ses voisins dans l’image. On a choisit la quatre connexité plutôt que la huit connexité donc. Le terme de droite permet de maintenir une cohérence entre l’image donnée Y en niveau de gris et l’image X qui est l’image sur laquelle travaille l’algorithme et que l’on récupère en sortie. Les poids et permettent d’ajuster l’importance que l’on donne aux deux termes. Le terme de droite mesure en fait la distance entre la valeur examinée et la valeur correspondante dans l’image donnée. Vaut 0 si la valeur examinée vaut 0. Le est une valeur récupérée dans une matrice servant à faire du ‘dithering’. Plus le est élevé et plus la distance sera élevée et donc plus l’énergie sera élevée et moins on aura de chance d’accepter la valeur. Le est nul si car celui-ci a une incidence uniquement si l’on souhaite passer le pixel en blanc. Le agit ainsi comme une restriction au passage du pixel en blanc.

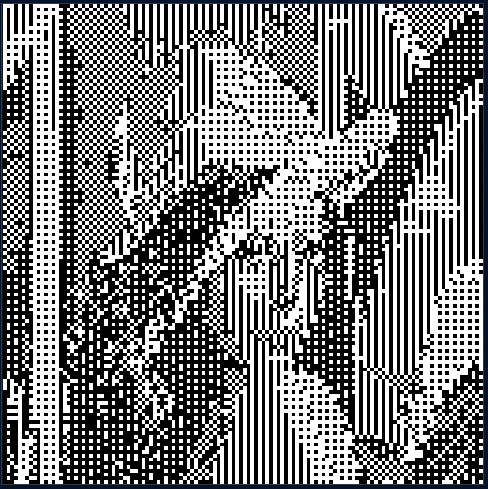
Contrairement au cas du débruitage d’image où le terme de régularisation avec les pixels voisins (clique d’ordre 2) est utile pour éliminer le bruit (on ne veut pas garder le bruit dans l’image bruitée Y), il nous a semblé qu’en implémentant du dithering avec notre méthode ce terme devenait inutile. On pourrait donc considérer une clique d’ordre 1 (auquel cas on a toujours et donc le terme est toujours nul) ou simplement fixer de façon à ne pas en tenir compte dans la fonction d’énergie. De cette manière on converge plus rapidement vers la solution.

La matrice de dithering est en fait appliquée à l’image en la translatant sur cette dernière. Cette translation sur l’image n’est pas effectuée directement mais en utilisant la fonction d’énergie présentée plus haut au sein de l’algorithme de Metropolis-Hastings. C’est cette matrice qui est responsable de l’effet artistique obtenue sur l’image de sortie. L’image de sortie, bien qu’étant en noir et blanc, ressemble à une image en niveau de gris lorsqu’on s’éloigne de l’écran ou que l’on regarde une version réduite de l’image. Pour savoir quel seuil appliquer à chaque pixel de l’image sélectionné aléatoirement, on utilise le reste de la division euclidienne avec la taille de la matrice de dithering. Plus la matrice de dithering est grande et plus on va pouvoir afficher des motifs variés. Par exemple avec une matrice de taille 2 par 2 on aura seulement cinq types de motifs (100% blanc, 75% blanc, 50% blanc, 25% blanc, 0% blanc).

Les matrices de bayer dont la taille doit être une puissance de 2 peuvent être construites récursivement en partant de la matrice de taille 2 par 2 grâce à la formule suivante :



Où est la matrice carré de Bayer de taille .



*Dithering avec matrice de Bayer de taille 2 par 2.*

*Dithering avec matrice de Bayer de taille 8 par 8.*

*Dithering avec matrice de Bayer de taille 4 par 4.*

Ces résultats ont été obtenus avec 1 million d’itérations et une température multipliée par 0.99999 à chaque itération. La température de départ est fixée à 4. On remarque qu’il n’y a pas beaucoup de différence entre l’image du milieu (4\*4) et celle de droite (8\*8). En regardant les images d’assez loin, nos yeux jouent en quelque sorte le rôle d’un filtre passe-bas en mélangeant les pixels se trouvant dans un même voisinage. On a l’impression que l’image est encore en niveau de gris.